COMP5423 NATURAL LANGUAGE PROCESSING Text Classification and Ranking

Assignment 1

Student Name: WU Yifan
Student ID: 24133101G

1. Method 1: Binary Classification

The core idea is to transform the Chinese word segmentation problem into a binary classification problem. For a given Chinese sentence, where each possible word boundary position is classified as either a word boundary or not, we can use a binary classification model to predict the word boundary position.

1.1 Tag definition for binary classification

Positive tag: the current position is a word boundary. Define as 1.
Negative tag: the current position is not a word boundary. Define as 0.

Example:

Sentence: 我/喜歡/蘋果
Tag Vector: [1, 0, 1, 0, 1]

1.2 Feature extraction for binary classification

For each character, we extract the following features:

- contextual features: Capture the contextual information of the two characters before and after.
  e.g. for Chinese character '喜', the contextual windows is [BOS, '我', '喜', '歡', '蘋']
- position features: the relative proportion of whole sentence is calculated.
  e.g. for Chinese character '喜', the proportion is 2/4 = 0.5
- character features: DictVectorizer encoding or One-Hot encoding.
  e.g '喜' can be encoded using one-hot or DictVectorizer.

After this, a whole feature vector can be written as:

(contextual features, character features, position features) -> (positive tag/negative tag)

e.g. Chinese character '喜' in sentence '我/喜歡/蘋果' can be constructed as:

((BOS,1,2,3,4), 3, 0.5) -> (0) or (1)

1.3 Logistic Regression

Logical regression is a commonly used classification algorithm, which is mainly used to solve dichotomous classification problems. When the output value is > 0.5, it is predicted to be category 1. When the output value is ≤ 0.5, it is predicted to be category 0. It is suitable to use in the binary classification of segmentation of Chinese sentence.

Here is a simple python code

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.linear_model import LogisticRegression
# feature generate (using DictVectorizer)
def generate_features(sentence):
    chars = list(sentence)
    features_list = []
    for i in range(len(chars)):
        features = {
            'c': chars[i],
            'c-1': chars[i-1] if i>0 else 'BOS',
            'c+1': chars[i+1] if i<len(chars)-1 else 'EOS',
            'c-1+c': (chars[i-1] if i>0 else 'BOS') + chars[i],
            'c+c+1': chars[i] + (chars[i+1] if i<len(chars)-1 else 'EOS'),
            'bigram': (chars[i-1] if i>0 else 'BOS') + chars[i],
            'trigram': (chars[i-2] if i>1 else 'BOS') + (chars[i-1] if i>0 else
'BOS') + chars[i],
            'pos': i/len(chars),
            'is_punct': 1 if chars[i] in {', ','。','! '} else 0
        }
        features_list.append(features)
    return features_list

# data preprocess
def preprocess_train_data(train_data):
    X, y = [], []
    for sentence in train_data:
        raw_sentence = sentence.replace(" ", "")
        labels = [0] * len(raw_sentence)

        ptr = 0
        for word in sentence.split():
            word_len = len(word)
            if ptr + word_len - 1 < len(raw_sentence):
                labels[ptr + word_len - 1] = 1
            ptr += word_len

        features = generate_features(raw_sentence)
        X.extend(features)
        y.extend(labels)
    return X, y

# train data
train_data = [
    "我 愛 吃 蘋果",
    "我 愛 吃 香蕉",
    "西瓜 和 梨 也 是 不错 的 選擇"
]

# split
X_train, y_train = preprocess_train_data(train_data)
vectorizer = DictVectorizer(sparse=False)
X_train_vec = vectorizer.fit_transform(X_train)
```

```
# training
model = LogisticRegression(
    penalty='l2',
    C=0.1,
    solver='lbfgs',
    max_iter=1000,
    class_weight='balanced'
)
model.fit(X_train_vec, y_train)

train_pred = model.predict(X_train_vec)

# Testing
def predict(sentence):
    chars = list(sentence)
    features = generate_features(sentence)
    X_vec = vectorizer.transform(features)
    preds = model.predict(X_vec)

    result = []
    for i, char in enumerate(chars):
        result.append(char)
        if i < len(preds) and preds[i] == 1:
            result.append(' ')
    return ''.join(result).strip().replace('  ', ' ')

# 测试
test_data = ["我愛吃西瓜和蘋果，香蕉也是不错的選擇，梨我不愛吃。"]
for sent in test_data:
    print(f"\nOriginal Sentence: {sent}")
    print(f"Segmentation Result: {predict(sent)}")
```

```
Original Sentence: 我愛吃西瓜和蘋果，香蕉也是不错的選擇，梨我不愛吃。
Segmentation Result: 我 愛 吃 西瓜 和 蘋果 ， 香蕉 也 是 不错 的 選擇 ， 梨 我 不 愛 吃 。
```

2. Method 2: Multi-Classification

2.1 BMSE labelling system

BMES labels each character of a Chinese sentence with one of the following four labels:

- B (Begin): The beginning character of the word.
- M (Middle): denotes the middle character of the word
- E (End): Indicates the end character of a word
- S (Single): Indicates a single character in a word

e.g. Sentence: 我熱愛自然語言處理 BMSE: SBEBMMMME

2.2 Features extraction

- contextual features: Capture the contextual information of the two characters before and after.
  e.g. for Chinese character '愛', the contextual windows is ['我', '熱', '愛', '自', '然'].
- position features: the relative proportion of whole sentence is calculated.
  e.g. for Chinese character '愛', the proportion is 3/9 = 0.33
- character features: DictVectorizer encoding or One-Hot encoding.
  e.g '愛' can be encoded using one-hot or DictVectorizer.

((1,2,3,4,5), 3, 0.33) -> (B) (M) (S) or (E)

The way to differentiate between multiple classifications is that the model needs to learn which words are combined to give the highest probability of E, the end-of-word identity, so that the classification model trained for word segmentation.

2.3 Random Forest

Random forests are suitable for BMES labelling tasks because they naturally support multi-classified problems (B/M/E/S), can handle complex contextual feature relationships.

Here is a simple Python code

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction import DictVectorizer

# features generate
def prepare_data(sentences):
    X, y = [], []
    for sent in sentences:
        words = sent.split()
        labels = []
        raw = []
        for word in words:
            raw.extend(list(word))
            if len(word) == 1:
                labels.append('S')
            else:
                labels.extend(['B'] + ['M']*(len(word)-2) + ['E'])

        for i in range(len(raw)):
            features = {
                'c-2': raw[i-2] if i>1 else '<S>',
                'c-1': raw[i-1] if i>0 else '<S>',
                'c': raw[i],
                'c+1': raw[i+1] if i<len(raw)-1 else '</S>',
                'c+2': raw[i+2] if i<len(raw)-2 else '</S>',
                'pos': i/max(len(raw), 1),
                'bigram': (raw[i-1] + raw[i]) if i>0 else '<START>'
            }
            X.append(features)
            y.append(labels[i])
```

```
    return X, y

# train data
train_data = [
    "人工智能 很 強大 ",
    "自然語言處理 是 人工智能 領域 學習 的 一 部分",
    "我 熱愛 人工智能"
]

vec = DictVectorizer(sparse=False)
X_train, y_train = prepare_data(train_data)
X_train = vec.fit_transform(X_train)

# training
model = RandomForestClassifier(
    n_estimators=100,
    max_depth=8,
    min_samples_split=5,
    class_weight='balanced'
)
model.fit(X_train, y_train)

# segmentation
def segment(sentence):
    chars = list(sentence)
    X_test = []
    for i in range(len(chars)):
        features = {
            'c-2': chars[i-2] if i>1 else '<S>',
            'c-1': chars[i-1] if i>0 else '<S>',
            'c': chars[i],
            'c+1': chars[i+1] if i<len(chars)-1 else '</S>',
            'c+2': chars[i+2] if i<len(chars)-2 else '</S>',
            'pos': i/max(len(chars), 1),
            'bigram': (chars[i-1] + chars[i]) if i>0 else '<START>'
        }
        X_test.append(features)

    X_test = vec.transform(X_test)
    labels = model.predict(X_test)

    result = []
    current = []
    for char, label in zip(chars, labels):
        current.append(char)
        if label in ['E', 'S']:
            result.append(''.join(current))
            current = []

    return ' '.join([word for word in result if word.strip()])
# testing
test_sentence = "我熱愛自然語言處理，學習人工智能，人工智能領域很強大"
print("\nOriginal Sentence:", test_sentence)
print("Segmentation Result:", segment(test_sentence))
```

```
Original Sentence: 我熱愛自然語言處理，學習人工智能，人工智能領域很強大
Segmentation Result: 我 熱愛 自然語言處理 ， 學習 人工智能 ， 人工智能 領域 很 強 大
```